Matthew Fischer

Assignment 4

Computer Programming

Summary of four chapters on Prolog

The answer is terms, and there are four kinds of term in Prolog: atoms, numbers, variables, and complex terms (or structures).

Atoms are strings of unbroken sequences of characters that make up the rules and facts names. Atoms are also single quote strings ‘ ‘. The last is special character strings. Such as: ; , :- . Theses have pre-defining meanings that create the logic in the database. Spaces are a different type of character. It is hidden shows the end of an Atom. Numbers are not as important in typical Prolog Applications. Float numbers are used and integers for counting lists of elements. Variables are upper and lower case letters and numbers and can include underscores. Complex terms use the building blocks of the other three to make the facts and rules. Complex terms are also called structures.

Syntax of Prolog separates, ends, and declares the facts, rules and queries. Period means “a full stop.” The semi-colon means “or”. You can imply something biased on two different facts can be true. Comma means “and.” The “:-“ means if second then the first is true;

Three Basic Constructs

Three basic constructs in Prolog. The Facts and Rules are a collection that make up the knowledge base or the data base. Prolog is all about making the database of information and for the third construct, Query. Query is to request the information in the database.

Facts and Rules are declaring information in the database. Another import link to the fact is the “.” period. The period represents “a full stop.” If there is no period than the query can never be read.

Facts are when something is declared as a true. Statements ex: “happy(mia).” Mia is happy. Rules are when using Facts you can infer some other fact. listensToMusic(mia) :- happy(mia). The rule is when mia is happy. She listens to music.

Queries start with a “?-“ which ends with a period. It will appear like a method. Using the name of the Fact or Rule as the name of the Question requesting with parenthesis and a variable name or an object inside. A query can answer yes or no when asking a question about a rule. Or the query returns a name when asking for all X.

Modus Ponens

Modus Ponens is when a rule is stated with a “head :- body” or more commonly known as an if statement. If the body is true than the head is true.

This solved all the core basics of the language. This establishes the how to understand the language.

Exercise 1.3

Atoms: 1,2,3,4,5,9

Variables:6,8,10

Neither: 7

Exercise 1.2

Atoms: 2

Variables:

Complex Terms: 1,3,4,5,6,7

Neither: 8,9,10

Exercise 1.3

-vincent is a woman

-mia is a woman

-jules is a man

Man and women are persons

If father than loves

If y is a man and son(z,y) than father is (y,z)

If man y and son(z,y) than father is (y,z)

Exercise 1.4

Killer(Butch).

Married(Mia,Marsellus).

dead(Zed).

Loves(Mia) := goodDancer(x).

Eats(Jules) := nutritious(x) , tasty(x).

Exercise 1.5

1.yes

2.no

3.no

4. no

5. no

6.Y = ron

Y = harry

7. no

The chapter one practical programs taught how to connect to a database and how to use the listing query.

Chapter 2

Unification, and proof search

Unification is the way you unify two terms together. Unification is separated by “,” a comma. Unification can be done by simple variables or complex constructs. It compares and checks if they are equal. By using the equal sign to check if they are equal.

Unification can happen several ways. First: On a knowledge base the query will search the database for an atom matching the atom, then are unified if they are the same. Second if the atom is an unknown variable and the other is an assigned atom then the they unify. Third: in comparing complex terms must have the same name. and they mush have the same number of arguments. Then it searches the database to see if there are like terms and to see if it can unify. Fourth is list recursion. When the list uses recursion to send the tail back through to see if there is a match to the list in question.

Exercises 2.1

1. bread  =  bread - unify
2. ’Bread’  =  bread - if term 2 is instantiated to term 1
3. ’bread’  =  bread - unify
4. Bread  =  bread - if term 1 is instantiated to term 2
5. bread  =  sausage - if term 1 is instantiated to term 2
6. food(bread)  =  bread - unify
7. food(bread)  =  X - unify
8. food(X)  =  food(bread) - if term 1 is instantiated to term 2
9. food(bread,X)  =  food(Y,sausage) - unify
10. food(bread,X,beer)  =  food(Y,sausage,X) - unify
11. food(bread,X,beer)  =  food(Y,kahuna\_burger) - unify
12. food(X)  =  X - unify
13. meal(food(bread),drink(beer))  =  meal(X,Y) - if term 1 is instantiated to term 2
14. meal(food(bread),X)  =  meal(X,drink(beer)) - unify

Exercises 2.2

1. satisfied
2. harry = wizard(X)
3. satisfied
4. satisfied
5. Herminone = hermione

Exercises 2.3

|  |
| --- |
| a criminal eats a big kahuna burger |
| a criminal eats every criminal |
| a criminal eats every big kahuna burger |
| a criminal likes a criminal |
| a criminal likes a big kahuna burger |
| a criminal likes every criminal |
| a criminal likes every big kahuna burger |
| a big kahuna burger eats a criminal |
| a big kahuna burger eats a big kahuna burger |
| a big kahuna burger eats every criminal |
| a big kahuna burger eats every big kahuna burger |
| a big kahuna burger likes a criminal |
| a big kahuna burger likes a big kahuna burger |
| a big kahuna burger likes every criminal |
| a big kahuna burger likes every big kahuna burger |
| every criminal eats a criminal |
| every criminal eats a big kahuna burger |
| every criminal eats every criminal |
| every criminal eats every big kahuna burger |
| every criminal likes a criminal |
| every criminal likes a big kahuna burger |
| every criminal likes every criminal |
| every criminal likes every big kahuna burger |
| every big kahuna burger eats a criminal |
| every big kahuna burger eats a big kahuna burger |
| every big kahuna burger eats every criminal |
| every big kahuna burger eats every big kahuna burger |
| every big kahuna burger likes a criminal |
| every big kahuna burger likes a big kahuna burger |
| every big kahuna burger likes every criminal |
| every big kahuna burger likes every big kahuna burger |

Exercise 2.4

|  |
| --- |
| word(V1, \_,\_11,\_,\_21,\_,\_31,\_), |
| word(V2, \_,\_12,\_,\_22,\_,\_32,\_), |
| word(V3, \_,\_13,\_,\_23,\_,\_33,\_), |
| word(H1, \_,\_11,\_,\_12,\_,\_13,\_), |
| word(H2, \_,\_21,\_,\_22,\_,\_23,\_), |
| word(H3, \_,\_31,\_,\_32,\_,\_33,\_). |

Chapter 3

Recursion when checking a database in a tree. Needs to bases to go to the predecessor and to the predecessor’s other children.

Exercise 3.1

Yes, the would be unable to resolve false assumptions. It would fail with an error ‘out of local stack’

Exercise 3.2

|  |
| --- |
| directlyIn(olga, natasha). |
| directlyIn(natasha, irina). |
|  |
| in(X, Y) :- |
| directlyIn(X, Y). |
| in(X, Y) :- |
| directlyIn(X, Z), |
| in(Z, Y). |

Exercise 3.3

travelFromTo(X,Y) :-

|  |
| --- |
| directTrain(X, Y). |
| travelFromTo(X, Y) :- |
| directTrain(X, Z), |
| travelFromTo(Z, Y). |

Exercise 3.4

|  |
| --- |
| greater\_than(succ(X), succ(Y)) :- |
| greater\_than(X, Y). |

Chapter 4

Lists

Lists with the H|T makes a easy way to cycle through recursively. The tail is its own list and you can take the head of the tail and the tail of the tail through the next recursive call.

In the query you check if the list atom matches the database for matches.

Exercise 4.1

|  |
| --- |
| 1. [a,b,c,d] = [a,[b,c,d]]. (false)  2. [a,b,c,d] = [a|[b,c,d]]. (true) |
| 3. [a,b,c,d] = [a,b,[c,d]]. (false) |
| 4. [a,b,c,d] = [a,b|[c,d]]. (true) |
| 5. [a,b,c,d] = [a,b,c,[d]]. (false) |
| 6. [a,b,c,d] = [a,b,c|[d]]. (true) |
| 7. [a,b,c,d] = [a,b,c,d,[]]. (false) |
| 8. [a,b,c,d] = [a,b,c,d|[]]. (true) |
| 9. [] = \_. (true) |
| 10. [] = [\_]. (false) |
| 11. [] = [\_|[]]. (false) |

Exercise 4.2

1. [1][2,3,4]] (correct list, 4 elements)

|  |
| --- |
| 2. [1,2,3|[]] (correct list, 3 elements) |
| 3. [1|2,3,4] (syntax error) |
| 4. [1|[2|[3|[4]]]] (correct list, 4 elements) |
| 5. [1,2,3,4|[]] (correct list, 4 elements) |
| 6. [[]|[]] (correct list, 1 element) |
| 7. [[1,2]|4] (syntax error) |
| 8. [[1,2],[3,4]|[5,6,7]] (correct list, 5 elements) |

Exercise 4.3

second(X, [\_,X|\_]).

Exercise 4.4

swap12([X,Y|T], [Y,X|T]).

Exercise 4.5

|  |
| --- |
| listtran([X|T0], [Y|T1]) :- |
| tran(X, Y), |
| listtran(T0, T1). |